UNITED STATES PATENT APPLICATION

SPARSE REFRESH OF DISPLAY

INVENTOR Thomas Willis   Citizenship: **USA**   Residence: **Mountain View, California**

INVENTOR Steven Midford   Citizenship: **USA**   Residence: **Portland, Oregon**

SPARSE REFRESH OF DISPLAY

FIELD

This invention relates generally to display devices and more particularly to

5   displaying information on a display device.

BACKGROUND

10   Current systems use raster-based display refresh techniques to update their

displays. Using this technique, a host electronic device transfers the entire displayed

contents to the display device at a fixed rate, which is often called the "refresh rate" and is

typically 60-85 Hz in prior systems. Each transfer typically moves a frame, also called a

screen image, from the host to the display device by tracing the screen image from left-to-

15   right and top-to-bottom on the display screen. This refresh is wasteful unless substantial

portions of the screen image change at approximately the refresh rate.

For example, consider a user reading a news story from a web page. The content

of the displayed page changes only as the user scrolls through the story; yet, current

systems built on raster-refresh techniques expend energy and effort to repeatedly copy the

20   same data, i.e., the screen contents, from the host to the display. This repeated copying

inefficiently uses power and bandwidth between the host and the display.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 depicts a block diagram of a electronic device for implementing an

embodiment of the invention.

25   Fig. 2 depicts a block diagram of an example frame buffer using addressing by

regions, according to an embodiment of the invention.

Fig. 3 depicts a flowchart of example processing for a write command, according to an embodiment of the invention.

Fig. 4 depicts a block diagram of example snoop logic, according to an embodiment of the invention.

5   Fig. 5 depicts a flowchart of example processing for a graphics engine, according to an embodiment of the invention.

Fig. 6 depicts a flowchart of example processing for a graphics engine, according to an embodiment of the invention.

DETAILED DESCRIPTION

10   Fig. 1 depicts a block diagram of an electronic device 100 for implementing an embodiment of the invention. The electronic device 100 may include a graphics engine 105, a snoop logic 110, a frame buffer 115, a sparse-refresh scan-out logic 125, and a display device 130. Although the graphics engine 105, the snoop logic 110, the frame buffer 115, the sparse-refresh scan-out logic 125, and the display device 130 are drawn as 15   being separate entities, in other embodiments some or all may be packaged together. The electronic device 100 may be implemented using any suitable hardware and/or software, such as a personal computer available from a number of vendors. But, other examples may be portable computers, network computers, laptop or notebook computers, PDAs (Personal Digital Assistants), mainframe computers, or any other suitable electronic 20   devices.

The hardware and software depicted in Fig. 1 may vary for specific applications and may include more or fewer elements than those depicted. For example, other peripheral devices such as audio adapters, or chip programming devices, such as EPROM (Erasable Programmable Read-Only Memory) programming devices may be used in 25   addition to or in place of the hardware already depicted. Thus, an embodiment of the invention may apply to any hardware configuration that supports displaying information on a display device.

The graphics engine 105 generates graphics or text for display on the display device 130. In an embodiment, the graphics engine 105 may be implemented as hardware logic. In another embodiment, the graphics engine 105 may be implemented as instructions within memory that are executable on a processor. Although the graphics

5   engine 105 is drawn as being part of the electronic device 100, in another embodiment the graphics engine 105 may be external to the electronic device 100. The graphics engine 105 may be connected to the snoop logic 110 and the frame buffer 115. The graphics engine 105 may include a memory 107, in which write commands to the buffer 115 are buffered. Although the memory 107 is shown contained within the graphics engine 105,

10   in another embodiment the memory 107 is external to the graphics engine 105, either internal to the electronic device 100 or external to the electronic device 100. In another embodiment, the memory 107 is not present or not used. The functions of the graphics engine 105 are further described below with reference to Fig. 5.

Referring again to Fig. 1, the snoop logic 110 identifies modified regions of the

15   frame buffer 115 and tracks regions that have not yet been sent to the display device 130. The snoop logic 110 may be communicatively coupled to the graphics engine 105, the frame buffer 115, and the sparse-refresh scan-out logic 125. Portions of the snoop logic 110 are further described below with reference to Figs. 3 and 4. In an embodiment, the snoop logic 110 may be implemented as hardware logic. In another embodiment, the

20   snoop logic 110 may be implemented as instructions within memory that are executable on a processor. Although the snoop logic 110 is drawn as being a part of the electronic device 100, in another embodiment the snoop logic 110 may be external to the electronic device 100.

The frame buffer 115 is a region of memory that holds the image to be displayed

25   on the display device 130. The frame buffer 115 may be comprised of a single plane that contains data for all color components to be displayed on the display device 130 or may be comprised of independent planes that each contain the data for one color component. In another embodiment, the frame buffer 115 may contain values that are indexes into a table of colors. In other embodiments, other organizations of the frame buffer 115 may be

used. The frame buffer 115 may be local to a graphics sub-system or may be shared with other agents. In an embodiment, the frame buffer 115 may be implemented as an adapter card. The frame buffer 115 may be communicatively coupled to the graphics engine 105, the snoop logic 110, and the sparse-refresh scan-out logic 125. Example contents of the

5   frame buffer 115 are further described below with reference to Fig. 2.

Referring again to Fig. 1, the sparse-refresh scan-out logic 125 writes a selected region or regions within the frame buffer 115 to the display device 130 when so instructed by the snoop logic 110. The sparse-refresh scan-out logic 125 writes the selected region or regions to the display device 130 asynchronously from the writes from the graphics

10   engine 105 to the frame buffer 115. The sparse-refresh scan-out logic 125 may include instructions in memory capable of being executed on a processor to carry out the functions of the present invention. In another embodiment, some or all of the functions of the sparse-refresh scan-out logic 125 may be carried out via hardware in lieu of a processor-based system. The sparse-refresh scan-out logic 125 may be communicatively

15   coupled to the snoop logic 110, the frame buffer 115, and the display device 130.

The sparse-refresh scan-out logic 125 uses requests from the snoop logic 110 to drive the scan-out operation. In an embodiment, sparse-refresh the scan-out logic 125 may replace raster-based logic used in graphics controllers. The sparse-refresh scan-out logic 125 may copy the region specified by the snoop logic 110 from the frame buffer 115

20   to the display device 130. The sparse-refresh scan-out logic 125 may format pixels from the frame buffer 115 in the appropriate format for presentation on the display device 130. Because a modified region is sent to the display device 130 asynchronously to graphics operations, a region might be further modified while the sparse-refresh scan-out logic 125 is copying the region to the display device 130. Depending on whether or not the newly

25   modified pixels have been scanned out, these modifications may or may not be reflected in the information that the sparse-refresh scan-out logic 125 sends to the display device 130.

To address this potential problem, in one embodiment the sparse-refresh scan-out logic 125 may have the highest priority access to the frame buffer 115. The highest priority access may be implemented by a number of techniques. In an embodiment, the highest priority access is implemented by the graphics engine 105 holding off all write

5   operations to the frame buffer 115 while the sparse-refresh scan-out logic 125 is reading the frame buffer 115. In an embodiment, the graphics engine 105 buffers write operations while the sparse-refresh scan-out logic 125 is reading the frame buffer 115. In another embodiment, the graphics engine 105 detects which region that the sparse-refresh scan-out logic 125 is reading from the frame buffer 115 and only buffers those writes directed

10   to that region while allowing write operations to other regions within the frame buffer 115 to proceed. These functions of the graphics engine 105 are further described below with reference to Fig. 5.

The display device 130 communicates information to the user of the electronic device 100. The display device 130 may be communicatively coupled to sparse-refresh

15   the scan-out logic 125. The display device 130 may be a cathode-ray tube (CRT) based video display well known in the art of computer hardware. But, in other embodiments the display device 130 may be replaced with a liquid crystal display (LCD) based or gas, plasma-based, flat-panel display. In still other embodiments, any appropriate display device may be used. Although only one display device 130 is shown, in other

20   embodiments, any number of display devices of different types or of the same type may be present. Although the display device 130 is drawn as part of the electronic device 100, in other embodiments the display device 130 may be external to the electronic device 100.

As will be described in detail below, aspects of an embodiment pertain to specific apparatus and method elements implementable on electronic devices. In another

25   embodiment, the invention may be implemented as a program product for use with a electronic device. The programs and data structures defining the embodiment may be delivered to a electronic device via a variety of signal-bearing media, which include, but are not limited to:

(1) information permanently stored on a non-rewriteable storage medium (e.g., read-only memory devices attached to or within a electronic device, such as a CD-ROM readable by a CD-ROM drive, or any other type of non-rewriteable storage medium);

(2) alterable information stored on a rewriteable storage medium (e.g., a hard disk,
5   diskette, tape, random-access memory device, or any other type of rewriteable storage medium); or

(3) information conveyed to a electronic device by a communications medium, such as through a computer or telephone network accessed via a network adapter, including wireless communications.

10   Such signal-bearing media, when carrying processor-readable instructions that direct the functions of the present invention and/or data organized in a data structure, represent embodiments of the present invention.

Fig. 2 depicts a block diagram of example contents of the frame buffer 115 using a region addressing technique, according to an embodiment of the invention. Example
15   contents of the frame buffer 115 show an addressing scheme in which the screen of the display device 130 is divided into a number of x-pixel by y-pixel regions. The pixels for each region are packed into the frame buffer 115 using linear addressing. That is, the first xy pixels in the frame buffer 115 correspond to the first area; the second xy pixels correspond to the adjacent area, and so forth.

20   Fig. 2 illustrates addressing for a 4-pixel square frame buffer without padding between regions. But, in another embodiment, there may be padding between rows within an area and between areas. Each square corresponds to a pixel on the display device 130, and the number in each square indicates the position in the frame buffer 115. The solid lines indicate the region boundaries and the dashed lines indicate pixel boundaries with
25   the regions being 2- pixels square. Thus in the example shown, region 0 (255) contains pixels 0, 1, 2, and 3; region 1 (260) contains pixels 4, 5, 6, and 7; region 2 (265) contains pixels 8, 9, 10, and 11; and region 3 (270) contains pixels 12, 13, 14, and 15. In another

embodiment, the pixels may be represented in another order within the regions. An organization of regions in the frame buffer 115 provides better locality in the reference stream for the frame buffer 115 for most drawing operations. Giving the 2-dimensional spatial coherence typical of screen drawing, it is desirable to locate nearby pixels, in both

5   the horizontal and vertical directions, as close as possible within the frame buffer.

Although the example of Fig. 2 is drawn to contain four regions with four pixels each, in other embodiments any number of regions and pixels may be present. Although the example of Fig. 2 is drawn with regions having a square shape, in other embodiments any region shapes may be used. Although the example of Fig. 2 is drawn with contiguous

10   regions, in other embodiments, the regions need not be contiguous. Although Fig. 2 is drawn with each of the regions having the same number of pixels, in other embodiments some or all of the regions may contain a different number of pixels. Although Fig. 2 is drawn with regions not overlapping, in another embodiment the regions may overlap.

Fig. 3 depicts a flowchart of example processing for the snoop logic 110,

15   according to an embodiment of the invention. Control begins at block 300. Control then continues to block 303 where the snoop logic 110 initializes the last modified region to be none. The last modified region is a variable that will be used later in the processing of Fig. 3. Control then continues to block 305 where the snoop logic 110 detects a write operation from the graphics engine 105 to the frame buffer 115.

20   Control then continues to block 310 where the snoop logic 110 determines the region associated with the frame buffer write command. In an embodiment, the snoop logic 110 determines the region by using a lookup table that maps a frame buffer address to a region number. In another embodiment, the snoop logic 110 determines the region using a logical transformation. Control then continues to block 315 where the snoop logic

25   110 determines whether the region previously determined at block 310 is the same as the last modified region. If the determination at block 315 is true, then control returns to block 305, as previously described above.

If the determination at block 315 is false, then control continues to block 320 where the snoop logic 110 instructs the sparse-refresh scan-out logic 125 to send the region from the frame buffer 115 to the display 130. Control then continues to block 325 where the snoop logic 110 sets the last modified region to be the region being written.

5   Control then returns to block 305, as previously described above.

Thus, as illustrated by Fig. 3, the snoop logic 110 causes the frame buffer 115 to accumulate writes by the graphics engine 105 to a first region in the frame buffer 115 until the graphics engine 105 writes to a different region in the frame buffer 115. When the graphics engine 105 writes to the different region, the snoop logic 110 causes the

10   sparse-refresh scan-out logic 125 to write the first region from the frame buffer 115 to the display device 130.

Fig. 4 depicts a block diagram of example logic 400 for implementing the snoop logic 110, according to an embodiment of the invention. The logic 400 may include D-type flip-flops 410, 420, and 430, and the compare logic 440. Using the standard

15   nomenclature for flip-flops in Fig. 4, "Q" indicates output, "D" indicates data, and a triangle symbol indicates a clock input.

The d-type flip-flop 410 receives WRN 450 as a data input. The WRN 450 may be the region number of the region currently being written by the graphics engine 105 to the frame buffer 115. The d-type flip-flop 410 receives the write 455 as a clock input.

20   The write 455 may be high when the snoop logic 110 detects that the write from the graphics engine 105 to the frame buffer 115 has occurred. Thus, the write 455 indicates when the WRN 450 is valid. The d-type flip-flop 410 produces Q output, which serves as data input to the d-type flip-flop 420 and the compare logic 440.

The d-type flip-flop 420 receives as data input the Q output of the d-type flip-flop

25   410. The d-type flip-flop 420 also receives a clock signal from the output of the compare logic 440. The d-type flip-flop 420 produces Q output of the LRN 460, which is the region number of the last modified region. The LRN 460 is input to the d-type flip-flop 430 and the compare logic 440.

The d-type flip-flop 430 receives as D input the Q output of the d-type flip-flop 420. The d-type flip-flop 430 also receives a clock signal from the write 455. The d-type flip-flop 430 produces as Q output the SRN 465, which is the region number of the region to be sent to the display device 130. The SRN 465 is input to the sparse-refresh scan-out logic 125.

5

The comparator 440 may be logic that receives as input the LRN 460 and the Q output of the d-type flip-flop 410 and determines whether the two input signals are equal. In an embodiment, the compare logic 440 may be implemented as an exclusive-or gate. But, in other embodiments any appropriate logic may be used. The comparator 440 produces as output the SCAN 470, which is input to the sparse-refresh scan-out logic 125 and indicates when the sparse-refresh scan-out logic 125 should send the region identified by the SRN 465 to the display device 130.

10

Fig. 5 depicts a flowchart of example processing for the graphics engine 105, according to an embodiment of the invention. Control begins at block 500. Control then proceeds to block 510 where the graphics engine 105 generates a frame buffer write command. The graphics engine 105 may generate a frame buffer write command using data from an external source (not shown). In another embodiment, the graphics engine 105 may generate a frame buffer write command using its own data.

15

Control then continues to block 520 where the graphics engine 105 detects whether the frame buffer 115 is currently being accessed by the sparse-refresh scan-out logic 125. If the determination at block 520 is true, then control continues to block 530 where the graphics engine 105 determines which region of the frame buffer 115 is being accessed by the sparse-refresh scan-out logic 125.

20

Control then continues to block 540 where the graphics engine 105 determines whether the region that the sparse-refresh scan-out logic 125 is accessing in the frame buffer 115 is the same as the region in the frame buffer write command, which was previously generated at block 510. If the determination at block 540 is true, then control continues to block 550 where the graphics engine 105 stores the write command in the

25

memory 107. Commands stored in the memory 107 are sent to the frame buffer 115 at a later time when the regions accessed in the buffered write command are not being accessed by the sparse-refresh scan-out logic 125.

If the determination at block 540 is false, then control continues to block 560 where the graphics engine 105 sends the write command to the frame buffer 115. Control then returns to block 510 as previously described above.

If the determination at block 520 is false, then control continues to block 560 where the graphics engine 105 sends the write command to the frame buffer 115. Control then returns to block 510 as previously described above.

Fig. 6 depicts a flowchart of processing for the graphics engine 105, according to an embodiment of the invention.

Control begins at block 600. Control then continues to block 605 where the graphics engine 105 determines whether there are any regions left to process in a set of candidate regions. The set of candidate regions may be selected according to a variety of criteria, including all regions in the frame buffer 115, all regions that have not been written to in a period of time, all regions except a number of most-recently written to regions, a number of least-recently written to regions, and all regions that are being displaced from the frame buffer 115. But, in another embodiment any appropriate selection criteria may be used.

If the determination at block 605 is false, then control continues to block 699 where the logic returns.

If the determination at block 605 is true, then control continues to block 615 where the graphics engine 105 determines whether the current region is dirty (modified). If the determination at block 615 is false, then control continues to block 645 where the graphics engine 105 moves the current region to the next region. Control then returns to block 605, as previously described below.

If the determination at block 615 is true, then control continues to block 620 where the graphics engine 105 copies the current region from the frame buffer 115 to the display device 130. Control then continues to block 625 where the graphics engine 105 determines whether the current region in the frame buffer 115 was written to during the

5   copy at block 620. If the determination at block 625 is false, then control continues to block 635 where the graphics engine 105 marks the current region to be clean, or unmodified. Control then continues to block 645, as previously described above.

If the determination at block 625 is true, then control continues to block 640 where the graphics engine 105 marks the current region to be dirty, or modified. The actions of

10   blocks 625 and 640 are necessary because modified regions may be sent to the display device 130 asynchronously to graphics operations, so it is possible that a region may be further modified while the graphics engine 105 is copying the region to the display device 130. Depending on whether or not the newly modified pixels have been scanned out, these modifications may or may not be reflected in the data sent to the display device 130.

15   As a result, the graphics engine 105 only marks a region as clean (unmodified) if there were no writes to the region during the scan-out process. Otherwise, the region is left marked dirty (modified). Control then continues to block 645, as previously described above.

In the previous detailed description of exemplary embodiments of the invention,

20   reference was made to the accompanying drawings (where like numbers represent like elements), which form a part hereof, and in which is shown by way of illustration specific exemplary embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, but other embodiments may be utilized and logical, mechanical, electrical, and other

25   changes may be made without departing from the scope of the present invention. The previous detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

Numerous specific details were set forth to provide a thorough understanding of the invention. However, the invention may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in detail in order not to obscure the invention.

5